Proyecto programado I – Haskell.

José Andrés Navarro Acuña.

2016254241.

Instituto Tecnológico de Costa Rica.

Ingenieria en computación.

IC-4700 Lenguajes de programación.

José Enrique Araya Monge.

Grupo 1.

I semestre 2018.

Indice

1. Introducción.

El presente apartado tiene la finalidad de presentar el problema planteado a resolver.
El proyecto consiste en implementar en el lenguaje de programación Haskell, un mecanismo para recalcular los valores de variables que dependen de otras variables. Se tendrán ecuaciones que establezcan la forma de calcular el valor de una variable como función de otras. Por ejemplo:

```
x = 2 + w
y = x - z
r = w * t
```

Como se puede observar, la variable "*y*" depende de la variable "*x*", por tanto, para el cálculo de "*y*" se debe calcular tambien "*x*".
La principal idea es ir almacenado ecuaciones conforme se van insertando, sin embargo, existen algunos casos donde no debe ser posible almacenar ecuaciones. El primer caso corresponde a que, una variable solo puede aparecer a lo sumo una vez en el lado izquierdo de una igualdad ya que, no se trata de resolver un sistema de ecuaciones, de modo que el siguiente caso es inválido:

```
r = w + t
r = 2 * w
```

Por tanto, una variable ya declarada no puede volver a declararse.
Un segundo caso corresponde a que la variable que aparece al lado izquierdo no puede aparecer al lado derecho, por ejemplo:

```
n = r * n
```

Otro caso importante corresponde a que, tampoco se puede dar la formación de ciclos de dependencias entre variables, porque de nuevo eso es plantear un sistema de ecuaciones matemáticas. En el siguiente caso la tercera ecuación forma el siguiente ciclo de dependencias: "*r*" depende de "*t* " que depende de "*s*" que depende de "*r*":

```
r =   w + t
t =   w * s
s =   q + r
```

En otras palabras las dependencias entre variables deben formar un grafo dirigido acíclico.
Debido a las dependencias que se pueden formar entre ecuaciones, conforme se van introduciendo nuevas ecuaciones se deben ir recalculando cuáles variables se necesitan para el cálculo y sus fórmulas correspondientes.

Por ejemplo, se introduce la siguiente ecuación en el programa: $y = a + b$, del cual contendrá la siguiente información:

variable	parámetros originales	ecuación original	parámetros vigentes	ecuación vigente
y	a, b	a + b	a, b	a + b

Si ahora se introduce la ecuación $x = y * z$ se debe utilizar la información ya almacenada sobre *y*, para cambiar la fórmula a $x = (a + b) * z$, ya que *y* aparece en la ecuación de "*x"*. Esto se especifica cambiando el conjunto de parámetros y la ecuación y llamándolos *vigentes:*

variable	parámetros originales	ecuación original	parámetros vigentes	ecuación vigente
y	a, b	a + b	a, b	a + b
x	y, z	y * z	~~y, z~~ a, b, z	~~y * z~~ (a + b) * z

Si se introduce la ecuación $b = 2 + c$, "*b"* no es afectada por "*x*" y "*y*" , (ninguna de ellas aparece en la ecuación de b); por otro lado "*x*", "*y*" son afectadas por "*b*" (ya que "*b*" aparece en sus ecuaciones vigentes). Por tanto, se debe usar la información de "*b*" para actualizar los parámetros y fórmulas vigentes de "*x*" y "*y*":

variable	parámetros originales	ecuación original	parámetros vigentes	ecuación vigente
y	a, b	a + b	~~a, b~~ a, c	~~a + b~~ a + (2 + c)
x	y, z	y * z	~~a, b, z~~ a, c, z	~~(a + b) * z~~ (a + (2 + c)) * z
b	c	2 + c	c	2 + c

Además, existe el caso donde si se inserta una nueva ecuación puede que ninguna de las ecuaciones definidas con anterioridad sufra cambios debido a la ecuación insertada, además, podría suceder el mismo caso pero al contrario, ninguna de la ecuaciones existentes podrían no afectar a la ecuación insertada.

Posteriormente, se presentan las funcionalidades que el programa debe de poseer.

- **Insertar ecuación (ie):**

Funcionalidad encargada de introducir una nueva ecuación en el ambiente.
Además, se deben de verificar que ninguna de las siguientes situaciones sucedan:

- La variable a la izquierda de la ecuación no debe tener una ecuación previa.

- La variable a la izquierda de la ecuación no debe aparecer a la derecha de esa misma ecuación.
- La nueva ecuación no debe formar un ciclo de dependencias con las ecuaciones anteriores; esto es para cada una de las ecuaciones anteriores, revisar que la variable de la nueva ecuación no aparece a la derecha de una ecuación anterior y que la variable de esa ecuación anterior no aparece a la derecha de la nueva ecuación; para esta revisión debe usar las ecuaciones vigentes.
- La ecuación se recibirá como una tira de texto que sigue el formato especificado arriba.
- Rechazar la ecuación si tiene errores sintácticos en ecuación.

Además, se debe de mostrar la información respectiva de la ecuación insertada, por ejemplo:

```
>> ie y = a + b
{ y, [a, b], a + b, [a, b], a + b }
```

- **Mostrar variable (mv):**

Dada una variable debe mostrar la información asociada con esa variable; en particular se muestra el conjunto de parámetros originales de esa variable y el conjunto de parámetros vigentes, por ejemplo:

```
>> mv y
{ y, [a, b], a + b, [a, b], a + b }
```

- **Mostrar ambiente (ma):**

Mostrar la información asociada con todas las variables que se han definido.

```
>> ma

{ y, [a, b], a + b, [a, c], a + (2 + c) }
{ x, [y, z], y * z, [a, c, z], (a + (2 + c)) * z }
{ b, [c], 2 + c, [c], 2 + c }
```

- **Calcular variable (cv):**

Dada una variable y un conjunto de valores correspondientes a sus parámetros vigentes, debe evaluar la ecuación que calcula dicha variable dados esos parámetros.

*Esto es: (12 + (2 + 30)) * 97*

```
>> cv x 12 30 97
4268

>> cv y 97 20
119
```

Esto es: 97 + (2 + 20)

- **Calcular variable original (cvo):**

Como el anterior caso, solo que deben usarse los parámetros originales.

*Esto es: 5 * 9*

```
>> cvo x 5 9
45

>> cvo y 5 32
37
```

Esto es: 5 + 32

- **Mostrar parámetros (mp):**

Esta funcionalida devuelve la lista de parámetros que se requieren para calcular todas las variables. Por ejemplo:

```
>> mp
a c z
```

- **Evaluar todo (et):**

Usa los valores especificados para los parámetros en el orden mostrado por la operación anterior, *mostrar-parámetros*, y evalúa todas las variables definidas por ecuaciones.

```
>> et 45 3 91
[ (y, 50) (x, 4550) (b, 5) ]
```

Esto es: a = 45, b = 3, z = 91

- **Terminar (fin):**

Simplemente terinar el ciclo de ejecución del programa.

2. Estructuras de datos usadas.

Seguidamente, se presentarán las estructuras utlizadas para la ejecución del programa.
Primero y principal se útilizó una estructura de datos "*Arbol*", cuyos constructuros de datos son "*Hoja*" y "*Nodo*", para el caso de "*Hoja*" se estableció que solo acepte otra estructura llamada "*Temino*", cuyos constructores de datos corresponden a "*Variable*", del cual acepta un valor *String,* y "*Entero*", del cual acepta un valor numérico entero. Continuando con la estructura "*Arbol*", el constructor "*Nodo*" solo acepta como parametros una operación seguido de dos arboles, de los cuales representan los nodos izquierdo y derecho. Con respecto a la operación, se estableció una estructura *type* del cual se le dió sinónimo al tipo de datos "*Int -> Int -> Int*" para que represente la operación aritmetica como tal.
Por otro lado, se estableció una estructura *type* del cual se le dio significado a "*Estado*", del cual tiene la forma de una lista que contiene una tupla con los siguientes parámetros: una variable *String*, una lista de *Strings*, una estructura "*Arbol*", otra lista de *Strings*, y finalmente, otra estructura "*Arbol*". Con respecto a los criterios utilizados en la estructura *Estado*, básicamente ninguno, la inserción de datos sigue el orden con conforme se insertan datos, es decir, no se sigue un patrón acorde a una variable.

3. Instrucciones para ejecutar el programa.

Como principal requerimiento para ejecutar el proyecto es necesario poseer en el ordenador el ambiente de Haskell, llamado *Haskell Plataform* en su versión 8.2.2. Seguidamente, se debe ejecutar el programa llamado *WinGHCi* en el cual se procederá a buscar y compilar el archivo con la extensión .hs del proyecto. Una vez compilado el proyecto el usuario ya tiene la posibilidad de acceder a las diversas funcionalidades que el programa ofrece.
Primeramente, para ejecutar el programa se debe ingresar en la linea de comandos la palabra "*main*" y presionar la tecla *Enter* como se muestra a continuación:

```
Prelude> :load "Proyecto.hs"
[1 of 1] Compiling Main             ( Proyecto.hs, interpreted )
Ok, one module loaded.
*Main> main

>>
```

Una vez completado este paso, ya es posible acceder a todos los comandos que el proyecto presenta. A continuación la explicación del uso de cada uno.

- **Insertar ecuación (ie):**

Para la inserción de una nueva ecuación en el ambiente, se debe escribir en la linea de comandos la frase "*ie*" seguida de la variable además del signo de asignación "=" separados por un espacio, posteriormente se inserta la ecuación con el siguiente formato "*a + b*", donde las únicas operaciones validas son la suma (+), la resta (-) y la multiplicación (*). Cabe destacar que la ecuación insertada debe seguir estrictamente el siguiente formato: *variable_1 operación variable_2*, cualquier otra ecuacion que no siga este formato será rechazada. Finalmente, se debe presionar la tecla *Enter* para ejecutar el comando. A continuación un ejemplo de ejecución.

```
>> ie x = a + b
{ x, [a, b], a + b, [a, b], a + b }

>> ie y = 6 - x
{ y, [x], 6 - x, [a, b], 6 - (a + b) }

>> ie i = y * x
{ i, [y, x], y * x, [a, b], (6 - (a + b)) * (a + b) }
```

- **Mostrar variable (mv):**

En caso de querer visualizar todos los datos respectivos a una ecuacion insertada con anterioridad, unicamente se debe insertar la frase "*mv*" separada de un espacio seguido de la incognita respectiva de la ecuación que se desea mostrar. El único parametro de entrada de este comando corresponde a dicha incógnita de la ecuación. Finalmente, se debe presionar la tecla *Enter* para ejecutar el comando. A continuación un ejemplo de ejecución.

```
>> mv x
{ x, [a, b], a + b, [a, b], a + b }
```

- **Mostrar ambiente (ma):**

La ejecución de dicho comando se realiza de manera sencilla, lo único que se debe ingresar en linea de comandos es la frase "*ma*" y seguidamente, se mostrarán todas las ecuaciones almacenadas hasta el momento. Finalmente, se debe presionar la tecla *Enter* para ejecutar el comando. A continuación un ejemplo de ejecución.

```
>> ma

{ x, [a, b], a + b, [a, b], a + b }
{ y, [x], 6 - x, [a, b], 6 - (a + b) }
{ i, [y, x], y * x, [a, b], (6 - (a + b)) * (a + b) }
```

- **Calcular variable (cv):**

En lo que respecta la ejecución de dicho comando, se debe escribir en la linea de comandos la frase "*cv*" seguido del nombre de alguna incoginita que se haya declarado con anterioridad, esto separado mediante un espacio, seguidamente, se deben ingresar los valores enteros necesarios para calcular el árbol vigente asociado a la ecuación ingresada, nuevamente, separados mediante espacios. Posteriormente, se mostrará el resultado de dicho calculo. Dichos valores enteros requeridos se pueden visualizar mediante el comando *Mostrar variable (mv).* Finalmente, se debe presionar la tecla *Enter* para ejecutar el comando. A continuación un ejemplo de ejecución.

```
>> mv i
{ i, [y, x], y * x, [a, b], (6 - (a + b)) * (a + b) }

>> cv i 12 7
-247
```

En este caso la variable "*a*" tomaría el valor 12 y la variable *" b"* el valor de 7.

- **Calcular variable original (cvo):**

Para la ejecución de este comando se debe ingresar la frase "*cvo*", donde posteriormente se debe ingresar la incognita relaciona a la ecuación separada por un espacio. Con este comando sucede algo similiar al anterior, una vez ingresada la incognita se deben ingresar los valores enteros relacionados al arbol original donde estos deberán ir separados por espacios. Nuevamente, estos valores requeridos pueden ser consultados mediante el comando *Mostrar variable (mv).* Finalmente, se debe presionar la tecla *Enter* para ejecutar el comando. A continuación un ejemplo de ejecución.

```
>> mv i
{ i, [y, x], y * x, [a, b], (6 - (a + b)) * (a + b) }

>> cvo i 12 7
84
```

En este punto, la variable "*y*" tomaria el valor 12 y la variable "*x*" el valor de 7. Luego se obtendría, en este caso, la multiplicación de ambos valores,

- **Mostrar parametros (mp):**

Para poder visualizar todas las variables de las ecuaciones definidas hasta el momento, se debe ingresar la frase "*mp*" en la linea de comandos, cabe destacar que este comando no recibe ningún tipo de parametros, por ende solo basta con insertar la frase anterior mencionada. Un comando que puede ser util al momento de visualizar los parametros del ambiente corresponde a *Mostrar ambiente (ma),* ya que se podrán visualizar las ecuaciones con sus respectivos arboles vigentes.
Finalmente, se debe presionar la tecla *Enter* para ejecutar el comando. A continuación un ejemplo de ejecución.

```
>> ma

{ x, [a, b], a + b, [a, b], a + b }
{ y, [x], 6 - x, [a, b], 6 - (a + b) }
{ i, [y, x], y * x, [a, b], (6 - (a + b)) * (a + b) }

>> mp
a b
```

En este caso, tanto las ecuaciones "*x*", "*y*" y "*i* " poseen las variables "*a*" y "*b*".

- **Evaluar todo (et):**

En lo que respecta a este comando, se evaluarán todas las ecuaciones con respecto a su árbol vigente. Primeramente, se debe ingresar en la linea de comandos la frase "*et*" seguido de una cantidad especifica de valores enteros de los cuales corresponden a la ejecución del comando anterior mencionado, *Mostrar parametros (mp).* Todos estos valores además del comando, deben estar separados por espacios debidamente. Finalmente, se debe presionar la tecla *Enter* para ejecutar el comando. A continuación un ejemplo de ejecución.

```
>> mp
a b

>> et 12 7
[ (x, 19) (y, -13) (i, -247) ]
```

Como es de intuir, la cantidad de variables que retorna la ejecución del comando *Mostrar parametros (mp)* será la cantidad de valores enteros que se deberán ingresar para el cálculo de este comando.

- **Finalizar ejecución del programa (fin):**

Para finalizar la ejecución del programa basta con ingresar en la linea de comandos la frase "fin" y posteriormente presionar la tecla *Enter*. A continuación un ejemplo de la ejecución.

```
>> fin
Saliendo...
*Main>
```

4. Corridas de ejemplo.

A continuación, se presenta una ejecución del código, donde se probaron todas y cada de las funcionalidades esperadas.

```
>> ie x1 = y + z
{ x1, [y, z], y + z, [y, z], y + z }

>> ie x1 = 1 + n
Error, la incógnita 'x1' ya se encuentra definida!.

>> ie x = x1 * w
{ x, [x1, w], x1 * w, [y, z, w], (y + z) * w }

>> mv x
{ x, [x1, w], x1 * w, [y, z, w], (y + z) * w }

>> cv x 2 3 4
20

>> ie u = w - t
{ u, [w, t], w - t, [w, t], w - t }

>> mp
y z w t

>> et 2 3 4 1
[ (x1, 5) (x, 20) (u, 3) ]

>> ie z = a * b
{ z, [a, b], a * b, [a, b], a * b }


>> ma

{ x1, [y, z], y + z, [y, a, b], y + (a * b) }
{ x, [x1, w], x1 * w, [y, a, b, w], (y + (a * b)) * w }
{ u, [w, t], w - t, [w, t], w - t }
{ z, [a, b], a * b, [a, b], a * b }

>> cv x 2 3 4 5
70
```

```
>> cvo x 2 3
6

>> ie w = 2 * u
Error, la incógnita 'w' produce un ciclo!.

>> ie w = 2 * z
{ w, [z], 2 * z, [a, b], 2 * (a * b) }

>> ma

{ x1, [y, z], y + z, [y, a, b], y + (a * b) }
{ x, [x1, w], x1 * w, [y, a, b], (y + (a * b)) * (2 * (a * b)) }
{ u, [w, t], w - t, [a, b, t], (2 * (a * b)) - t }
{ z, [a, b], a * b, [a, b], a * b }
{ w, [z], 2 * z, [a, b], 2 * (a * b) }

>> mp
y a b t

>> et 2 3 4 5
[ (x1, 14) (x, 336) (u, 19) (z, 12) (w, 24) ]

>> ie u = r * s
Error, la incógnita 'u' ya se encuentra definida!.

>> fin
Saliendo...
*Main>
```

5. Comentarios finales (estado del programa).

El estado final del presente proyecto es del 100%, es decir, todas las especificaciones planteadas requeridas fueron implementadas, entre ellas, los comandos, las validaciones de cada comando, la adaptación del ciclo iterativo proporcionado y cada cálculo que involucró cada comando para dar con el resultado respectivamente.

Con respecto a los problemas encontrados, basicamente ninguno, el único inconveniente fué enfrentarse a un lenguaje de programación en el cual nunca habia programado, por tanto, se requirió de tiempo para aprender e investigar de las logistica del funcionamiento de Haskell, por ejemplo, el recorrido de un árbol binario, las sustitución de nodos de arboles binarios, además de lista intencionales y el manejo de tuplas.

Finalmente, no se encontró ninguna limitación adicional que impidiera el desarrollo del proyecto.